CLAIMS

1. A method for memory management in execution of a program by a computer having a memory, comprising:

allocating respective portions of the memory to data objects using mutator threads of the program, whereby the objects are held in a heap created by the program;

tracing the data objects in the heap so as to mark the data objects that are reachable at a given stage in the program;

looping over the mutator threads so as to verify for each of the mutator threads that every update to the allocated portions of the memory in progress by the mutator thread has been completed; and

sweeping the heap so as to free the memory that is allocated to the data objects that are not marked as reachable, for reallocation to new data objects.

2. A method according to claim 1, wherein looping over the mutator threads comprises forcing each of the mutator threads to perform a fence operation.

3. A method according to claim 1, wherein to update the allocated portions of the memory, the mutator threads record pointer references, and wherein tracing comprises marking the data objects referenced by the pointer references as reachable.

4. A method according to claim 3, wherein looping comprises looping until all the data objects referenced by pointer references that have been newly discovered are marked as reachable.

5. A method according to claim 1, wherein looping over the mutator threads comprises verifying that each of the

mutator threads has completed performing every update in progress by ensuring that each of the mutator threads finishes a write barrier for every update it is executing.

6. A method according to claim 1, wherein tracing the data objects comprises tracing the objects using a collector thread.

7. A method according to claim 6, wherein looping over the mutator threads comprises verifying that each of the mutator threads has completed performing every update in progress by carrying out a handshake protocol between the collector and mutator threads.

8. A method according to claim 7, wherein carrying out the handshake protocol comprises sending an interrupt using the collector thread and handling the interrupt using the mutator threads.

9. A method according to claim 7, wherein carrying out the handshake protocol comprises setting a value of a status variable in the collector thread before looping over all the mutator threads, and testing the value of the status variable in the mutator threads against the value in the collector thread.

10. A method according to claim 9, wherein carrying out the handshake protocol comprises, after each one of the mutator threads has completed performing the updates, setting the value of the status variable in that one of the mutator threads equal to the value of the status variable in the collector thread.

11. A method according to claim 6, wherein the collector and mutator threads operate on the memory concurrently.

12. A method according to claim 1, wherein the memory is accessed in accordance with a relaxed-consistency architectural scheme.

13. Computing apparatus, comprising:

a memory, arranged to store data; and

one or more processors, coupled to allocate respective portions of the memory to data objects using mutator threads of a program running on the apparatus, whereby the objects are held in a heap created by the program, to trace the data objects in the heap so as to mark the data objects that are reachable at a given stage in the program, to loop over the mutator threads so as to verify for each of the mutator threads that every update to the allocated portions of the memory in progress by the mutator thread has been completed, and to sweep the heap so as to free the memory that is allocated to the data objects that are not marked as reachable, for reallocation to new data objects.

14. Apparatus according to claim 13, wherein the one or more processors are adapted to loop over the mutator threads so as to force each of the mutator threads to perform a fence operation.

15. Apparatus according to claim 13, wherein to update the allocated portions of the memory, the mutator threads record pointer references, and wherein the one or more processors are arranged to trace the data objects by marking the data objects referenced by the pointer references as reachable.

16. A method according to claim 15, wherein the one or more processors are arranged to loop until all the data

objects referenced by pointer references that have been newly discovered are marked as reachable.

17. Apparatus according to claim 13, wherein the one or more processors are arranged to verify that each of the mutator threads has completed performing every update in progress by ensuring that each of the mutator threads finishes a write barrier for every update it is executing.

18. Apparatus according to claim 13, wherein the one or more processors are arranged to trace the data objects using a collector thread.

19. Apparatus according to claim 18, wherein the one or more processors are arranged to verify that each of the mutator threads has completed performing every update in progress by carrying out a handshake protocol between the collector and mutator threads.

20. Apparatus according to claim 19, wherein the one or more processors are arranged to carry out the handshake protocol by sending an interrupt using the collector thread and handling the interrupt using the mutator threads.

21. Apparatus according to claim 19, wherein the one or more processors are arranged to carry out the handshake protocol by setting a value of a status variable in the collector thread before beginning to loop over all the mutator threads, and testing the value of the status variable in the mutator threads against the value in the collector thread.

22. Apparatus according to claim 21, wherein in accordance with the handshake protocol, after each one of

the mutator threads has completed performing the updates, the value of the status variable in that one of the mutator threads is set equal to the value of the status variable in the collector thread.

23. Apparatus according to claim 18, wherein the one or more processors are arranged so that the collector and mutator threads operate on the memory concurrently.

24. Apparatus according to claim 13, wherein the one or more processors are arranged to access the memory in accordance with a relaxed-consistency architectural scheme.

25. A computer software product, comprising a computer-readable medium in which code instructions are stored, which instructions, when read by a computer having a memory, cause the computer to allocate respective portions of the memory to data objects using mutator threads of a program in execution by the computer, whereby the objects are held in a heap created by the program, and further cause the computer to trace the data objects in the heap so as to mark the data objects that are reachable at a given stage in the program, to loop over the mutator threads so as to verify for each of the mutator threads that every update to the allocated portions of the memory in progress by the mutator thread has been completed, and to sweep the heap so as to free the memory that is allocated to the data objects that are not marked as reachable, for reallocation to new data objects.

26. A product according to claim 25, wherein the instructions cause the computer to loop over the mutator

threads so as to force each of the mutator threads to perform a fence operation.

27. A product according to claim 25, wherein to update the allocated portions of the memory, the mutator threads record pointer reference, and wherein the instructions cause the computer to trace the data objects by marking the data objects referenced by the pointer references as reachable.

28. A product according to claim 27, wherein the instructions cause the computer to loop until all the data objects referenced by pointer references that have been newly discovered are marked as reachable.

29. A product according to claim 25, wherein the instructions cause the computer to verify that each of the mutator threads has completed performing every update in progress by ensuring that each of the mutator threads finishes a write barrier for every update it is executing.

30. A product according to claim 25, wherein the instructions cause the computer to trace the data objects using a collector thread.

31. A product according to claim 30, wherein the instructions cause the computer to verify that each of the mutator threads has completed performing every update in progress by carrying out a handshake protocol between the collector and mutator threads.

32. A product according to claim 31, wherein the instructions cause the computer to carry out the handshake protocol by sending an interrupt using the

collector thread and handling the interrupt using the mutator threads.

33. A product according to claim 31, wherein the instructions cause the computer to carry out the handshake protocol by setting a value of a status variable in the collector thread before beginning to loop over all the mutator threads, and testing the value of the status variable in the mutator threads against the value in the collector thread.

34. A product according to claim 33, wherein in accordance with the handshake protocol, after each one of the mutator threads has completed performing the updates, the value of the status variable in that one of the mutator threads is set equal to the value of the status variable in the collector thread.

35. A product according to claim 30, wherein the instructions cause the collector and mutator threads to operate on the memory concurrently.

36. A product according to claim 25, wherein the instructions cause the computer to access the memory in accordance with a relaxed-consistency architectural scheme.